Orchestration & System Design — A Mini Chapter

Why Orchestration Matters

Projects become **systems** when they run repeatedly, serve others, and must be dependable. Orchestration is the discipline of **coordinating tasks, dependencies, and timing** so that data flows predictably from sources to decisions.

From Notebooks to Pipelines

A notebook is exploratory; a pipeline is **repeatable**. Moving from cell-by-cell to **task-by-task** requires:

- **Explicit inputs/outputs** (files, tables, APIs)
- **Idempotency** (safe to re-run)
- **Observability** (logging, metrics)
- **Separation of concerns** (config vs. logic vs. output)

DAGs (Directed Acyclic Graphs)

A DAG makes dependencies **explicit**. Nodes are tasks; edges define order. Acyclicity allows **topological sorting**, partial re-runs, and targeted retries. Even without Airflow/Prefect, the *concept* is crucial.

Reliability Patterns

- **Retries with backoff:** transient failures recover automatically.
- **Checkpoints:** persist intermediate artifacts to avoid recomputation.
- **Fail fast & alert:** surface errors early; log contextual details.
- **Idempotent writes:** write new artifact versions or overwrite deterministically.

Right-Sizing the Solution

Not every project needs a heavy scheduler. A thoughtful **folder layout** and one or two scripts often deliver 80% of the value:

```
/config        # parameters, env, secrets (never hardcode)
/src           # reusable functions
/scripts       # CLI wrappers / job entrypoints
/jobs          # human- or machine-triggered schedules
/data, /logs, /reports
```

Financial Engineering Example

A stock forecaster system might include:

1. **Ingest** daily prices (input: API/CSV; output: `prices_raw.json`)
2. **Clean** and validate (output: `prices_clean.json`)
3. **Model** (output: `model.json` with params/metrics)

4. **Report** or publish signals (output: `report.txt`)

Failure points: missing data day, schema drift, model divergence. Add **logging** at step boundaries and **checkpoint** artifacts.

Looking Ahead

This course closes with concepts; deeper dives (Airflow, Prefect, streaming systems, container orchestration) live in future electives. You should leave able to **map** your project to a pipeline and **explain** what would be needed to scale it.

Self-Check

- Can you draw your project as a DAG?
- Do you know where to log and checkpoint?
- Which steps would you automate first, and why?